

APPENDIX A

```
package jungle.vdbms.wdk.visual;
import java.io.*;
5  import java.awt.*;
import java.util.*;

/**
10  * <b> WrapperModel Class </b>
*
* <p>This class is used to model wrappers. A wrapper is
* identified as a set of operations, some of which are connected
* by links. Supported functions are:
* <ul>
15  * <li> construction of wrappers by adding or deleting links and
operations. </li>
* <li> rendering wrappers on a canvas using the WrapperView class.
</li>
* <li> response to user input using the WrapperController class
20  * <li> shifting all links and operations. </li>
* <li> scaling operations around center of gravity. </li>
* </ul>
* <p>Copyright (c) 1997, Jungle Corp.
* @title WrapperModel
25  * @author Stephan Erickson
* @version 1.0
*
*/

30  public class WrapperModel
    implements Serializable
    {

        //////////////// Constructor ////////////////

35  /** Create a wrapper model, and make it the current wrapper. */
    public WrapperModel() {
    }

        //////////////// Methods ////////////////

    /** Get vector of wrapper listeners */
    public Vector getWrapperListeners()
    {
40  return wrapperListeners;
    }

    /** Allow a Visual GUI to register interest in events. */
    public void addWrapperListener(WrapperListener wl)
50  {
        wrapperListeners.addElement(wl);
    }

    /** Get id */
    public String getId()
55  {
        return theId;
    }
}
```

```

5      /** Set the id **/
      public void setId(String id)
      {
          theId = id;
      }

      /** Get root operation **/
10     public Operation getRoot()
      {
          return root;
      }

      /** Set the root operation **/
15     public void setRoot(Operation op)
      {
          root = op;
      }

      /** Get Initialization File **/
20     public String getInitFile()
      {
          return theInitFile;
      }

      /** Set Initialization File **/
25     public void setInitFile(String initFile)
      {
          theInitFile = initFile;
      }

30     /** Add breakpoint to operation **/
      public void addBreakpoint(Operation op)
      {
35         op.isBreakpoint = true;
      }

      /** Remove breakpoint from operation **/
      public void removeBreakpoint(Operation op)
      {
40         op.isBreakpoint = false;
      }

      /** Does operation have breakpoint? **/
45     public boolean isBreakpoint(Operation op)
      {
          return op.isBreakpoint;
      }

      /** Warn message **/
50     public static void warn(Operation from, String message) {
          System.out.println("In operation " + from.getId() + ": " +
message);
      }

55     /** Warn message **/
      public static void warn(String message) {
          System.out.println(message);
      }

```

```

/** Print message */
public static void print(String message) {
    System.out.println(message);
}

//***** MODEL OPERATIONS *****

/**
 * Add operation to wrapper model
 * @param operation the operation to add
 * Does nothing if operation is null
 */

public void addOperation(Operation operation)
{
    if (operation!=null) {
        operation.setId(getNextOperationId());
        operation.setWrapperModel(this);
        if (!hasOperation(operation)) {
            theOperationVector.addElement(operation);
        }
    }
}

/**
 * Add operation to wrapper model, and group with
 * existing operation list
 * @param operation the operation to add
 * Does nothing if operation is null
 */

public void addOperation(Operation operation, OperationList
operationList)
{
    // Add operation to operation list
    if (operation!=null) {
        operationList.addOperation(operation);
        operation.setId(getNextOperationId());
        operation.setWrapperModel(this);
    }
    // Add operation list to wrapper
    if (operationList!=null) {
        if (operation!=null)
            operationList.setId(-operation.getId());
        operationList.setWrapperModel(this);
        if (!theOperationListVector.contains(operationList)) {
            theOperationListVector.addElement(operationList);
        }
    }
}

/**
 * Determine if operation is in wrapper
 * @param operation the operation
 * Returns false if operation is null
 */

public boolean hasOperation(Operation operation)
{

```

```

        if (operation!=null) {
            Enumeration e = theOperationVector.elements();
            for (;e.hasMoreElements();) {
                Operation v = (Operation)e.nextElement();
                if (v.getId()==operation.getId())
                    return true;
            }
        }
        return false;
    }

    /**
     * Remove Operation
     * @param operation the operation to remove
     * Does nothing if operation is null
     */
    public void removeOperation(Operation operation)
    {
        if (operation!=null) {
            removeLinks(operation);
            // Remove bound operation
            OperationList operationList = getOperationList(operation);
            if (operationList!=null) {
                operationList.removeOperation(operation);
                // If operation list is empty, remove list
                if (operationList.getOperations().size()==0) {
                    removeOperationList(operationList);
                }
            } else {
                // Remove free operation
                theOperationVector.removeElement(operation);
            }
        }
    }

    /**
     * retrieve operations associated with wrapper
     * @return vector of operations
     */
    public final Vector getOperations()
    {
        return theOperationVector;
    }

    /**
     * set operations associated with wrapper
     * @param vector of operations
     */
    public final void setOperations(Vector operations)
    {
        theOperationVector = operations;
    }

    /**
     * retrieve operation edge is pointing to
     * @return Operation

```

```

    */

    public final Operation getOperation(Operation operation, String
5    linkName)
    {
        Vector links = new Vector();
        Enumeration e = theLinkVector.elements();
        for (;e.hasMoreElements();) {
            Link link = (Link)e.nextElement();
10         if (link.getStartOperation()==operation) {
            if (link.getLabel().equals(linkName))
                return link.getEndOperation();
            }
        }
15     return null;
    }

    /**
20     * retrieve operation associated with id
     * @return Operation or null if no operations can be found
     */

    public final Operation getOperation(int id)
    {
25         Enumeration e = theOperationVector.elements();
        for (;e.hasMoreElements();) {
            Operation o = (Operation)e.nextElement();
            if (o.getId()==id) return o;
        }
30         Enumeration e2 = theOperationListVector.elements();
        for (;e2.hasMoreElements();) {
            OperationList ol = (OperationList)e2.nextElement();
            if (ol.getId()==id) return ol;
        }
35     return null;
    }

    /**
40     * Add Link
     * @param link the link to add.
     * Does nothing if link is null.
     */

    public void addLink(Link link)
45     {
        if (link!=null) {
            theLinkVector.addElement(link);
        }
    }

50

    /**
     * Remove Link
     * @param link the link to remove
     * Does nothing if link is null
55     */

    public void removeLink(Link link)
    {
        if (link!=null) {

```

```

        theLinkVector.removeElement(link);
    }
}

5  /**
   * Remove all links connected to a given operation
   * @param operation from which links should be removed
   * Does nothing if operation is null
   */

10 public void removeLinks(Operation operation)
    {
        if (operation!=null) {
            // Find links to delete
15         Vector deleteVector = new Vector();
            Enumeration e = theLinkVector.elements();
            for (;e.hasMoreElements();) {
                Link link = (Link)e.nextElement();
                if (link.getStartOperation()==operation ||
20                 link.getEndOperation()==operation) {
                    deleteVector.addElement(link);
                }
            }

25         // Delete list of links
            Enumeration e2 = deleteVector.elements();
            for (;e2.hasMoreElements();) {
                Link link = (Link)e2.nextElement();
                theLinkVector.removeElement(link);
30             }
        }
    }

35  /**
   * retrieve links associated with wrapper
   * @return vector of links
   */

40 public final Vector getLinks()
    {
        return theLinkVector;
    }

45  /**
   * set links associated with wrapper
   * @param vector of links
   */

50 public final void setLinks(Vector links)
    {
        theLinkVector = links;
    }

55  /**
   * retrieve links starting from operation
   * @return vector of links
   */

    public final Vector getLinks(Operation operation)

```

```

{
    Vector links = new Vector();
    Enumeration e = theLinkVector.elements();
    for (;e.hasMoreElements();) {
5        Link link = (Link)e.nextElement();
        if (link.getStartOperation()==operation ||
            link.getEndOperation()==operation) {
            links.addElement(link);
        }
10    }
    return links;
}

/**
15    * Update link ids after unserialization process
    * @param vector of links
    */

    public final void updateLinkIds()
20    {
        Enumeration e = theLinkVector.elements();
        for (;e.hasMoreElements();) {
            Link link = (Link)e.nextElement();

25            // Check if link is valid
            String linkName = link.getLabel();
            Operation startOperation =
getOperation(link.getStartOperationId());
            boolean foundValidLink = false;
30            if (startOperation!=null) {
                String[] linkArray = startOperation.getLinkNames();
                for (int i=0; i<linkArray.length; i++) {
                    String validName = linkArray[i];
                    if (validName.equals(linkName)) {
35                        foundValidLink = true;
                    }
                }
            }

40            // Remove link if it is not valid
            if (!foundValidLink || startOperation==null) {
                removeLink(link);
            }

45            // Update operation links
            Enumeration e2 = theOperationVector.elements();
            for (;e2.hasMoreElements();) {
                Operation operation = (Operation)e2.nextElement();
                // Update wrapper model
50                operation.setWrapperModel(this);
                if (link.getStartOperationId() == operation.getId()) {
                    link.setStartOperation(operation);
                }
                if (link.getEndOperationId() == operation.getId()) {
55                    link.setEndOperation(operation);
                }
            }
            // Update operation list links
            Enumeration e3 = theOperationListVector.elements();

```

```

    for (;e3.hasMoreElements();) {
        OperationList operationList = (OperationList)e3.nextElement();
        // Update wrapper model
        operationList.setWrapperModel(this);
5       if (link.getStartOperationId() == operationList.getId()) {
            link.setStartOperation(operationList);
        }
        if (link.getEndOperationId() == operationList.getId()) {
10          link.setEndOperation(operationList);
        }
        // Update operation list element links
        Vector operations = operationList.getOperations();
        Enumeration e4 = operations.elements();
        for (;e4.hasMoreElements();) {
15          Operation operation = (Operation)e4.nextElement();
            // Update wrapper model
            operation.setWrapperModel(this);
            if (link.getStartOperationId() == operation.getId()) {
20              link.setStartOperation(operation);
            }
            if (link.getEndOperationId() == operation.getId()) {
                link.setEndOperation(operation);
            }
        }
25    }
}

/**
30  * Update link ids after unserialization process
    * @param vector of links
    */

public final void updateRootOperation()
35 {
    Operation root = getOperation(1);
    setRoot(root);
    if (root!=null)
        root.setWrapperModel(this);
40 }

/**
    * Remove operation list
    * @param operationList the operation list to remove
45  * Does nothing if operation list is null
    */

public void removeOperationList(OperationList operationList)
50 {
    if (operationList!=null) {
        removeLinks(operationList);
        theOperationListVector.removeElement(operationList);
    }
}

55 /**
    * Return operation list containing operation
    * @param operation the operation
    * @return operation list containing operation or null

```



```

    */

public OperationList getOperationList(Operation operation)
{
    5      Enumeration e = getOperationLists().elements();
      for (;e.hasMoreElements();) {
          OperationList operationList = (OperationList)e.nextElement();
          Enumeration e2 = operationList.getOperations().elements();
          10      for (;e2.hasMoreElements();) {
              Operation v = (Operation)e2.nextElement();
              if (v==operation)
                  return operationList;
          }
      }
    15      return null;
}

/**
    20      * retrieve operation lists associated with wrapper
      * @return vector of links
    */

public final Vector getOperationLists()
{
    25      return theOperationListVector;
}

/**
    30      * set operation lists associated with wrapper
      * @param vector of operation lists
    */

public final void setOperationLists(Vector operationLists)
{
    35      theOperationListVector = operationLists;
}

/**
    40      * retrieve next unused operation id
      * @return operation id
    */

public int getNextOperationId()
{
    45      int i;
      for (i=1; i<Integer.MAX_VALUE; i++) {
          boolean cont = false;
          Enumeration e = theOperationVector.elements();
          Enumeration e2 = theOperationListVector.elements();
          50      for (;e.hasMoreElements();) {
              Operation operation = (Operation)e.nextElement();
              if (i==operation.getId())
                  cont = true;
          }
          55      for (;e2.hasMoreElements();) {
              OperationList operationList = (OperationList)e2.nextElement();
              Enumeration e3 = operationList.getOperations().elements();
              for (;e3.hasMoreElements();) {
                  Operation operation = (Operation)e3.nextElement();

```

```

        if (i==operation.getId())
            cont = true;
    }
    }
    if (!cont) break;
}
return i;
}

10 //***** SELECTION OPERATIONS *****

/**
 * Set current selected wrapper element
 * @param e new selected wrapper element
15 */

public void setSelectedElement(WrapperElement e)
{
    e.theSelectedElement = e;
20 }

/**
 * Return current selected wrapper element
 * @return wrapper Element
25 */

public WrapperElement getSelectedElement()
{
    return WrapperElement.theSelectedElement;
30 }

/**
 * Is wrapper element selected?
 * @param e wrapper element
35 * @return true/false
 */

public boolean isSelectedElement(WrapperElement e)
{
40     return (e==WrapperElement.theSelectedElement);
}

//***** DATA *****

45 // The wrapper model in current use
private static WrapperModel theWrapper = null;

// The wrapper listeners
Vector wrapperListeners = new Vector();

50 // The root operation
Operation root;

// List of operations
55 private Vector theOperationVector = new Vector();

// List of operation lists
private Vector theOperationListVector = new Vector();

```

5

```
// List of links
private Vector theLinkVector = new Vector();

// The id of the wrapper
private String theId = "*thunk*";

// The initialization file
private String theInitFile = null;
}
```

APPENDIX B

```
package jungle.vdbms.wdk.visual;
import java.awt.*;
import java.io.*;
import com.sun.java.swing.*;

/**
 * <b> Wrapper Element Class </b>
 *
 * <p>A wrapper element is either an operation or an operation list,
or a link
 * between two operations. Wrapper elements have a label, color,
size, associated
 * wrapper model and information about whether they are selected or
not. They also
 * can have an associated image, and id.
 *
 * <p>Copyright (c) 1997, Jungle Corp.
 * @title GraphElement
 * @author Stephan Erickson
 * @version 1.0
 */

public class WrapperElement
    implements Serializable
{
    /**
     * Constructor
     */

    public WrapperElement()
    {
        /**
         * Set label
         * @param label set label of element
         */

        public void setLabel(String label)
        {
            theLabel = label;
            FontMetrics fm
Toolkit.getDefaultToolkit().getFontMetrics(theFont);
            theLabelWidth = fm.stringWidth(label);
            theLabelHeight = fm.getHeight();
        }

        /**
         * Get label
         * @return get label of element
         */

        public String getLabel()
        {
            return theLabel;
        }
    }
}
```

```

    }

    /**
5      * Set color
      * @param color new color of element
      */

    public void setColor(Color color)
10    {
        theColor = color;
    }

    /**
15     * Get color
      * @return current color of element
      */

    public Color getColor()
20    {
        return theColor;
    }

    /**
25     * Set selected color
      * @param color new selected color
      */

    public void setSelectedColor(Color color)
30    {
        theSelectedColor = color;
    }

    /**
35     * Get selected color
      * @return current selected color
      */

    public Color getSelectedColor()
40    {
        return theSelectedColor;
    }

    /**
45     * Set size
      * @param size set size of element
      * Do nothing if new size is below minimum allowed size.
      */

    public void setSize(int size)
50    {
        //if (size>theMinimumSize)
            theSize = (double) size;
    }

55    /**
      * Get size
      * @return get size of element
      */

```

```

5      public int getSize()
        {
            return (int) theSize;
        }

10     /**
        * Set selected
        * @param selected whether or not element is selected
        */

        public void setSelected(boolean selected)
        {
15            if (selected==true)
                theSelectedElement = this;
            else
                theSelectedElement = null;
        }

20     /**
        * Is selected
        * @return whether or not element is selected
        */

25     public boolean isSelected()
        {
            return (this==theSelectedElement);
        }

30     /**
        * Get font
        * @return get font of element
        */

35     public Font getFont()
        {
            return theFont;
        }

40     /**
        * Get label width
        * @return get label width of element
        */

45     public int getLabelWidth()
        {
            return theLabelWidth;
        }

50     /**
        * Get label height
        * @return get label height of element
        */

55     public int getLabelHeight()
        {
            return theLabelHeight;
        }

```

```

5  /**
   * Set id
   * @param id id of element
   */

   public void setId(int id) {
       theId = id;
       FontMetrics fm
Toolkit.getDefaultToolkit().getFontMetrics(theFont);
10  theIdWidth = fm.stringWidth(id+"");
       theIdHeight = fm.getHeight();
   }

15  /**
   * Get id
   * @return id of element
   */

   public int getId() {
20     return theId;
   }

   /**
25   * Get id width
   */

   public int getIdWidth()
   {
30     return theIdWidth;
   }

   /**
   * Get id height
   */
35  public int getIdHeight()
   {
       return theIdHeight;
   }
40

   /**
   * Set image icon
   */

45  public void setImageIcon(ImageIcon icon, int x, int y)
   {
       theImageIcon = icon;
       theXIcon = x;
       theYIcon = y;
50  }

   /**
   * Get image icon
   */
55  public ImageIcon getImageIcon()
   {
       return theImageIcon;
   }

```

```

5      /**
        * Get x position of icon
        */

10     public int getXIcon()
        {
            return theXIcon;
        }

15     /**
        * Get y position of icon
        */

        public int getYIcon()
        {
            return theYIcon;
        }

20     /**
        * Always associate every wrapper element with the
        * wrapper model it belongs to.
        * Set associated wrapper model.
        */

25     public void setWrapperModel(WrapperModel model) {
        theWrapperModel = model;
    }

30     /**
        * Get associated wrapper model
        */

        public WrapperModel getWrapperModel() {
35         return theWrapperModel;
        }

        //***** PRIVATE DATA *****

40         // The size of this element.
        public double theSize = 10;
        // The minimum size of this element.
        protected double theMinimumSize = 5;

45         // The label associated with this element.
        protected String theLabel = "";
        // The width of the label
        protected int theLabelWidth = 0;
        // The height of the label
50         protected int theLabelHeight = 0;

        // The id of this element.
        protected int theId = 0;
        // The width of the id
55         protected int theIdWidth = 0;
        // The height of the id
        protected int theIdHeight = 0;
        // Current id (static counter)
        protected static int theCurrentId = 0;

```



```

// Image Icon to display
protected ImageIcon theImageIcon = null;
5 // X position of image icon
protected int theXIcon = 0;
// Y position of image icon
protected int theYIcon = 0;

// The font associated with this element.
10 protected Font theFont = new Font("Arial", Font.PLAIN, 14);

// Color of this element
protected Color theColor = Color.black;
// The selected color of this element
15 protected Color theSelectedColor = Color.red;

// Selected element
protected static WrapperElement theSelectedElement = null;

20 // The wrapper model
protected WrapperModel theWrapperModel = null;
}

```

APPENDIX C

```
package jungle.vdbms.wdk.visual;
import jungle.vdbms.wdk.interpreter.*;
5 import jungle.vdbms.wdk.interpreter.Environment;
import java.lang.reflect.*;
import java.awt.*;
import java.io.*;
import java.net.*;
10 import java.util.*;
import com.sun.java.swing.*;
import jungle.vdbms.wdk.util.*;

/** An Operation is the basic unit of processing for Visual WDK
15 * wrappers. Each Operation has an <tt>call</tt> method which does
* the real work, and can call other Operations as
* needed. Operations use the <tt>setXXX</tt> and <tt>getXXX</tt>
* naming convention for properties that are set at design time.
* <p> As an Operation developer, you should <b>never do
20 op.call(state)</b>.
* Instead, you must always use the static method call(from, op,
state).
* The static method takes care of debugging output and visual trace
* display; it catches exceptions. Every call should return
25 * an Environment, <b>never return null</b> from a call method. */

/* Stephan Erickson: Added a number of functions & merged the
* Operation with the GraphModel vertex class. */

30 public abstract class Operation
    extends WrapperElement
    {
        /** Flow of control works by invoking an operation, that is, calling
        * the call method of the operation, passing in any necessary
        35 * parameters. This can be started when the user hits the "Run"
        * button (or maybe clicks on the root Operation). The call method
        * runs, and in most cases will call the call method of one of the
        * Operations it is wired up to, and perhaps call repeatedly, in a
        * loop. By default, this call should have the semantics of a normal
        40 * method call: it waits for the call to complete, and then returns
        * control to the parent call method. We could have the option of
        * running multiple invocations in parallel, but mostly I think that
        * would add to confusion, so let's leave it out (but not forget
        * about it) for the first version. So we have a model where we
        45 * start at the top, and call methods depth-first as we traverse
        * the web site, and simultaneously work through the wrapper. */

        /**
        * Constructor
        50 */

        public Operation()
        {
            theCurrentId++;
            theId = theCurrentId;
            theSize = 15;
            name = shortName(this.getClass().getName());
        }
    }
}
```

```

/***** CALL METHODS *****/

/**
5  * Every operation must support an implementation of this method
  * @param state the start runtime state
  * @return state the end runtime state
  */

protected abstract Environment call(Environment state) throws
10 Throwable;

/**
  * This static method is what you should call, (within the body of
  * a non-static Operation.call method) to call another operation.
15  * That is, do <tt>call(this, getOp(), state)</tt>, not
  * <tt>getOp().call(state)</tt>. The difference is that this static
  * method will gracefully handle a null operation, and it also
  * keeps track of trace information and updates the visual display
  * (if there is one).
20  */

public static Environment call(Operation from, Operation op,
                               Environment state) throws Throwable {
  if (op == null || from.callsDisabled) return state;
25  signal("call", null, state, from, op);
  Environment result = state;
  try {
    result = op.call(state); // <== Real work here
  } catch (Exception e) {
30    // Deal with recovering from exception here. ???
    throw e;
  }
  signal("return", null, result, op, from);
  return result;
35 }

/***** LINKING OPERATIONS TOGETHER *****/

/**
40  * Every operation must provide the allowable link names via this
  function
  * @return list of allowable operation to operation link names
  */
45

public abstract String[] getLinkNames();

/**
  * This function is provided to make access to the link names
50  easier
  * @return vector of operation link names
  */

public Vector getLinkNamesVector() {
55  String[] linkNames = getLinkNames();
  Vector rval = new Vector();
  for (int i=0; i<linkNames.length; i++) {
    rval.addElement(linkNames[i]);
  }
}

```

```

        return rval;
    }

    /**
5       * Use this method to access the destination Operation of a link.
Links are
        * no longer stored in the operations themselves, but in the
wrapper model.
        * This allows us to avoid writing logic to keep two wrapper
10       * representations in sync.
        */

    public Operation getOperation(String linkName)
    {
15       Operation op = theWrapperModel.getOperation(this, linkName);
        return op;
    }

    /***** SIGNAL EVENTS *****/

20    /**
        * Tell any listeners that this event occurred. Listeners register
        * via WrapperModel.getWrapper().addWrapperListener(cl).
        */

25    public static void signal(String eventType, Object data,
                                Environment state, Operation from,
                                Operation to) {

        if (from!=null) {
30           WrapperModel wrapper = from.getWrapperModel();
           Vector listeners = wrapper.getWrapperListeners();
           for (int i = 0; i < listeners.size(); i++) {
               ((WrapperListener)listeners.elementAt(i))
35               .eventOccurred(eventType, data, state, from, to);
           }
        }
    }

    /***** COORDINATE OPERATIONS *****/

40    /**
        * Move operation to absolute coordinates x,y
        * @param x destination x coordinate
        * @param y destination y coordinate
45       */

    public void moveTo(int x,int y)
    {
50       x0 = x;
        y0 = y;
    }

    /**
55       * Move operation relative to coordinates x0,y0
        * @param deltaX relative x coordinate
        * @param deltaY relative y coordinate
        */

    public void moveBy(int deltaX,int deltaY) {

```

```

        x0+=deltaX;
        y0+=deltaY;
    }

5    /**
     * Scale size of operation using x as multiplier
     * @param x scale multiplier
     */

10   public void scale(double x) {
        if (theSize*x>theMinimumSize) {
            x0 *= x;
            y0 *= x;
        }
15   }

    /**
     * Return distance from x,y to operation
     * @param x the x coordinate
     * @param y the y coordinate
20   */

    public int distanceFrom(int x,int y) {
        return (int) Math.sqrt((x-x0)*(x-x0)+(y-y0)*(y-y0));
25   }

    /**
     * Follow link from url. Specify post query if POST
     * is required, otherwise specify null for the
     * postQuery parameter
     * @param url in string form
     * @param post query in string form (EG: a=34&b=this is a test&c=0)
35   */

    public String followLink(String url, String postQuery)
    {
        URL inUrl = null;
        String text = "";
        try {
            inUrl = new URL(url);
            text = netClient.getText(inUrl);
        } catch(Exception e) {
            e.printStackTrace();
            // SFE change
        }
        return text;
50   }

    public URL convertUrl(String url)
    {
        URL newUrl = null;
        try {
            newUrl = new URL(url);
        } catch(Exception ex) {
            ex.printStackTrace(); // SFE change
        }
        return newUrl;
55   }

```

```

    }

    /***** PROPERTY SETTER/GETTERS *****/

5    /**
     * The name of an Operation is a short label for use by the human,
     * and for debugging output. It defaults to the unqualified class
name,
     * but can be set to anything you want.
10    */

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

15    /**
     * The comment of an Operation can be a longish comment block
     * describing what this operation is used for in this wrapper.
     * There will also be Help documentation that says what each
Operation
20    * class is for, and how to use it. But that comes later.
     */

    public String getComment() { return comment; }
    public void setComment(String desc) { this.comment = desc; }

25    /**
     * The coordinates of the operation
     */

30    public void setX(int x) { x0 = (double)x; }
    public int getX() { return (int)x0; }
    public void setY(int y) { y0 = (double)y; }
    public int getY() { return (int)y0; }

35    /**
     * Given a qualified class name as argument, return the unqualified
part.
     * That is, the part after the last ".". Also remove "Operation"
if it is
40    * at the end.
     */

    public static String shortName(String className) {
45        if (className.endsWith("Operation"))
            className = className.substring(0, className.length()-9);
        int dot = className.lastIndexOf('.');
        if (dot == -1) return className;
        else return className.substring(dot+1);
    }

50    /**
     * Return the operation's name.
     */

55    public String toString() { return getName(); }

    /***** DATA *****/

```

```

    /** Name of operation */
    String name = "";
    /** Comment associated with operation */
    String comment = "";
5
    /** Should we print information for debugging WrapperBuilder? This
    is
        * for a developer debugging WrapperBuilder; not a user debugging
        * a wrapper. */
10    static boolean debugging = true;

    // The net client static object used to support followLink
    static Web netClient = new Web();

15    /** Is this operation a breakpoint? */
    public boolean isBreakpoint = false;
    /** Are calls disabled? */
    public boolean callsDisabled = false;

20    /* x coordinate of operation center */
    public double x0;
    /* y coordinate of operation center */
    public double y0;
}

```

APPENDIX D

```
package jungle.vdbms.wdk.operations;
import jungle.vdbms.wdk.visual.*;
5 import jungle.vdbms.wdk.visualgui.*;
import jungle.vdbms.wdk.interpreter.*;
import jungle.vdbms.wdk.interpreter.Environment;
import jungle.vdbms.wdk.util.*;
10 import java.util.*;
import java.net.*;

/**
 * Match Operation
 */
15 public class MatchOperation extends Operation {

    /**
     * Constructor
     */
20 public MatchOperation() {}

    /**
     * Iterate over text using the match expression entered
     * in the Match property.
     *
     * Additional options are:
     *   - Read initial text from URL (URL property)
     *   - Follow URLs after match (FollowLinks property)
     */
30 public final Environment call(Environment state)
    throws Throwable
35 {
    // Create new copy of state
    /*
    Environment newState = new Environment();
    state.transfer(newState);
40 */

    // Temporarily make mutable to debug memory usage. SFE change
    Environment newState = state;
    Interpreter interpreter = new Interpreter(newState);
45

    // Fetch url, input text, start position, end position
    String parentUrl = state.evaluateVariable(Environment.URL);
    String input = state.evaluateVariable(Environment.TEXT);
    String startStr = state.evaluateVariable(Environment.START);
    String endStr = state.evaluateVariable(Environment.END);
50 int startInt = -1;
    int endInt = -1;

    // Obtain start position
    if (startStr!=null) {
55 Integer startInteger = new Integer(startStr);
        startInt = startInteger.intValue();
    }
    // Obtain end position
```



```

if (endStr!=null) {
    Integer endInteger = new Integer(endStr);
    endInt = endInteger.intValue();
}
5   input = input.substring(startInt, endInt);

    // If start url specified
    if (!startUrl.equals("")) {
10      input = followLink(startUrl, null);
    }

    // Initialize input & match expression
    newState.setVariable(Environment.TEXT, input, false);

15   Vector vars = interpreter.getVariables(match);

    // Iteratively match
    for (;;) {
20      Vector expressions = interpreter.parseExpressionSequence(match);
      interpreter.evaluateExpressionSequence(expressions);
      if (!interpreter.hasMoreInput()) break;

      // Call link for each variable
      Enumeration e = vars.elements();
25      for (;e.hasMoreElements();) {
          String variable = (String)e.nextElement();
          String value = newState.evaluateVariable(variable);

          // If followLinks true, follow URLs
          if (followLinks.booleanValue()) {
30              TextParser parser = new TextParser(value);
              // Iterate through each url
              for (;parser.hasMoreUrls();) {
                  String url = parser.nextUrl(parentUrl);
35                  if (url==null) continue;
                  // Eliminate duplicates
                  if (theTable.get(url)==null) {
                      theTable.put(url, "");
                  } else {
40                      continue;
                  }

                  String newPage = followLink(url, null);
                  newState.setVariable(Environment.TEXT, newPage, false);
45                  newState.setVariable(Environment.URL, url, false);
                  newState.setVariable(Environment.START, ""+0, false);
                  newState.setVariable(Environment.END, ""+newPage.length(),
false);

                  /* Environment tmpState = */ newState = call(this,
50      getOperation(variable), newState);
                  /* tmpState.transfer(state); SFE change */
              }
          } else {
              // If followLinks false, do not follow URLs
55      int start = 0;
              int end = 0;
              try {
                  start = interpreter.getMatchStart(variable);
                  end = interpreter.getMatchEnd(variable);

```

```

    } catch (Exception ex) {
        // No selection on failure
    }
    newState.setVariable(Environment.TEXT, input, false);
5    newState.setVariable(Environment.START, ""+start, false);
    newState.setVariable(Environment.END, ""+end, false);
    /* Environment tmpState = */ newState = call(this,
    getOperation(variable), newState);
    /* tmpState.transfer(state); SFE change */
10    }
    }
    // Emit rows if necessary
    if (!emitRows.equals("")) {
15        interpreter.evaluateExpression("emit-rows('"+emitRows+"')", 0);
    }
    return state;
}

20 /** Return set of link names used by this operation */
    public String[] getLinkNames()
    {
        Environment env = new Environment();
        Interpreter interpreter = new Interpreter(env);
25        Vector linkVector = interpreter.getVariables(match);
        String[] linkArray = new String[linkVector.size()];
        linkVector.copyInto((String[])linkArray);
        return linkArray;
    }
30
    /////////////// Methods to define Properties ///////////////////

    String match = "";
    public String getMatch() { return match; }
35    public void setMatch(String match) { this.match = match; }

    String startUrl = "";
    public String getStartUrl() { return startUrl; }
    public void setStartUrl(String url) { this.startUrl = url; }
40

    Boolean followLinks = new Boolean(false);
    public Boolean getFollowLinks() { return followLinks; }
    public void setFollowLinks(Boolean followLinks) { this.followLinks =
45    followLinks; }

    String emitRows = new String("");
    public String getEmitRows() { return emitRows; }
    public void setEmitRows(String emitRows) { this.emitRows = emitRows;
50    }

    // Eliminate duplicates
    public static Hashtable theTable;
}

```

APPENDIX E

```
package jungle.vdbms.wdk.visual;
import java.awt.*;
import java.io.*;

5  /**
   * <b> Link Class </b>
   *
10  * <p>This class is used to represent links in a wrapper model.
   * A link is identified as a directed line connecting two operations.
   * A link is associated with: <br>
   * <ul>
   *   <li> A start operation </li>
15  *   <li> An end operation </li>
   * </ul>
   * <p>Copyright (c) 1997, Jungle Corp.
   * @title Link
   * @author Stephan Erickson
20  * @version 1.0
   */

public class Link
25  extends WrapperElement
   implements Serializable
   {

   //***** MODEL OPERATIONS *****

30  public Link()
   {
       super();
   }

35  /**
   * Constructor.
   * Provide start and end operations. Default constructor
   * returns a directed link with arrow pointing to end operation.
   * Does nothing if either operation is null
40  * @param start the start operation
   * @param end the end operation
   */

   public Link(Operation start, Operation end)
45  {
       theStartOperation = start;
       theStartOperationId = start.getId();
       theEndOperation = end;
       theEndOperationId = end.getId();
50  }

   /**
   * Return start operation of link
   * @return Start Operation
55  */

   public Operation getStartOperation()
   {
       return theStartOperation;
   }
}
```

```

    }

    /**
5      * Set start operation of link
      * @param Start Operation
      */

    public void setStartOperation(Operation start)
10    {
        theStartOperation = start;
        theStartOperationId = start.getId();
    }

    /**
15      * Set end operation of link
      * @param end Operation
      */

    public void setEndOperation(Operation end)
20    {
        theEndOperation = end;
        theEndOperationId = end.getId();
    }

    /**
25      * Return end operation of link
      * @return End Operation
      */

    public Operation getEndOperation()
30    {
        return theEndOperation;
    }

    /**
35      * Set start operation id of link
      */

    public void setStartOperationId(int id)
40    {
        theStartOperationId = id;
    }

    /**
45      * Return start operation id of link
      * @return Start Operation Id
      */

    public int getStartOperationId()
50    {
        return theStartOperationId;
    }

    /**
55      * Set end operation id of link
      */

    public void setEndOperationId(int id)

```

```

    {
        theEndOperationId = id;
    }

5    /**
    * Return end operation id of link
    * @return End Operation Id
    */

10   public int getEndOperationId()
    {
        return theEndOperationId;
    }

15   //***** GRAPHICS OPERATIONS *****

    /**
    * Distance from link
    * @param x the x coordinate
20   * @param y the y coordinate
    * @return distance from link
    */

    public double distanceFrom(double x, double y)
25   {
        double length =
            Math.sqrt((theEndOperation.x0-theStartOperation.x0)*
                      (theEndOperation.x0-theStartOperation.x0)+
                      (theEndOperation.y0-theStartOperation.y0)*
30   (theEndOperation.y0-theStartOperation.y0));
        double cos = (theEndOperation.x0-theStartOperation.x0)/length;
        double sin = (theEndOperation.y0-theStartOperation.y0)/length;
        double distance = Math.abs((theStartOperation.x0-x)*sin+
35   (y-theStartOperation.y0)*cos);
        return distance;
    }

    //***** PRIVATE VARIABLES *****

40   /* The start operation */
    private Operation theStartOperation;

    /* The end operation */
    private Operation theEndOperation;

45   /* The start operation id */
    private int theStartOperationId;

    /* The end operation id */
50   private int theEndOperationId;
}

```